Monitoring Robot Actions for Error Detection and Recovery

M. Gini and R. Smith
University of Minnesota
Minneapolis, MN 55455

ABSTRACT

Reliability is a serious problem in computer controlled robot systems. Although robots serve successfully in relatively simple applications such as painting and spot welding, their potential in areas such as automated assembly is hampered by programming problems. A program for assembling parts may be logically correct, execute correctly on a simulator, and even execute correctly on a robot most of the time, yet still fail unexpectedly in the face of real world uncertainties. Recovery from such errors is far more complicated than recovery from simple controller errors, since even expected errors can often manifest themselves in unexpected ways. In this paper we present a novel approach for improving robot reliability. Instead of anticipating errors, we use knowledge-based programming techniques so that the robot can autonomously exploit knowledge about its task and environment to detect and recover from failures. We describe preliminary experiment of a system that we have designed and constructed in our Robotics Lab.

1. INTRODUCTION

We want to make robots more dependable so that they can be trusted when left unattended. This paper describes the design and development of a robot system that continues to operate satisfactorily even after it encounters a serious error [Gin83b], [Gin85c].

Failures in achieving a task are the result of errors, but not every error produces an immediately detectable failure. Errors can occur at many levels, at the mechanical level (a joint becomes locked), at the hardware level (a sensor does not function properly so that the robot is driven to exceed its joint limits), at the controller level, in the computer controlling the robot (either at the hardware or the software level), and in the environment. We are mostly interested in errors in the environment because they tend to be more unpredictable and difficult to characterize with mathematical models. We are interested in errors in the component parts used for the assembly, and in errors in the work cell (loaders, feeders, conveyor belts, tools). Our goal is to automatically detect problems caused by collisions, jammed parts, gripper slip, misorientation, alignment errors, and missing parts.

In practice, robot systems that can recover from errors without human intervention do not exist today because robot control programs cannot handle the vast range of possible error conditions. It takes uncommon skill and experience to develop such a program, and the resulting program will then only apply to the specific robot task at hand. Moreover, the program may have to be largely rewritten to handle even a minor change in the robot's task [Car85], [Loz83].

A difficult problem in automatic error detection and recovery is detecting that something significant has occurred. Many events are usually reported to the robot controller but not all of them are significant. The same event may be important in some circumstances and almost irrelevant in others. Deciding when something is important is the first step in the error detection process. The second step involves detecting the cause of the error and its effect on the robot environment. Errors might appear a long time after what caused them happened making it more difficult to detect them. Some errors do not affect the execution of the task so they could be left unrecovered. Only after the cause for the error has been identified or, at least, after alternative plausible causes have been found the recovery activity can start.

The robot system discussed here is geared towards industrial assembly tasks. The assembly to be performed is described in a robotics language. If an error occurs while the robot is performing the task, the robot detects the error, and dynamically plans the steps it must take to recover. To do this, the robot applies general knowledge about robots, and assembly tasks, plus specific information about the robot and the program in question.

Our approach simplifies robot programming by putting the burden for general error recovery on the robot system itself. The programmer can concentrate on the task at hand and minimize later maintenance if the robot recovers from most errors itself. This saves engineering time as well as robot downtime.

The system described here works in conjunction with an existing robot programming language. We show later in the paper how we have used it with an implementation of the AML language for the IBM 7565 robot. In addition to developing the testbed with the IBM robot we have developed prototype components of a simulated system. The simulated version is currently more complete than the testbed version since it is much easier to control a simulated robot than a real one.

2. INTELLIGENT ROBOT ERROR RECOVERY

Our system operates in two phases: offline, and online. Figure 1 illustrates its structure. Each box represents a separate program that may run as an independent process. The Preprocessor prepares the assembly task for execution by the robot by generating an Augmented Program (AP). The AP Executive interprets the robot program and monitors the robot's operation. If a serious error occurs, it activates the Recoverer. The Recoverer examines information from the event trace and from the task knowledge base and devises a recovery plan. More details on the component of the system are provided later in the paper.

Since the Preprocessor and the Recoverer both rely on symbolic computational techniques and do not require real-time performance they both reside on the same processor, referred to here as the Manager. Robot control operations require real-time response and reside on other processors.

Design Features of the Testbed

The design of the error recovery testbed incorporates four features of particular interest. First, the automated reasoning of the Manager and real-time functions of the AP Executive operate independently and execute on separate processors. Second, sensors are activated and monitored selectively according to the robot's current action. Third, sensor data are evaluated and assigned qualitative meanings at several levels

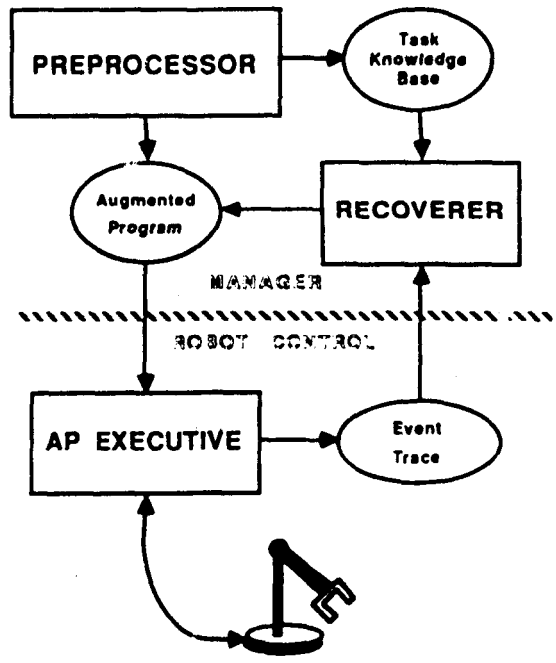

Figure 1: Error Recovery System Components

of the system. Fourth, if the robot's task fails repeatedly, the AP Executive will handle successive restarts and recoveries without ill effect.

The present configuration of the error recovery testbed manages an IBM 7565 manufacturing manipulator. The components of the error recovery Manager exist on an LMI Lambda processor. Motion control and sensor filtering for the IBM 7565 is implemented on an IBM Series 1 minicomputer using the AML programming system [Tay82]. The AP Executive resides on an MC68000 based personal computer system, an Apple Macintosh.

The IBM 7565 is well suited for these experiments. It is a cartesian robot moving on linear tracks over a rectangular work cell. The gripper has six degrees of freedom and its jaw contains six strain gauges. The 7565 is provided with the AML robot programming system which can be used to develop relatively sophisticated programs. AML provides facilities for monitoring the robot sensors and for performing robot motion subject to the presence of appropriate sensor readings.

Separation of reasoning and real-time

The testbed utilizes separate processors for executing the reactive, or real-time, software components and for executing the reflective, or symbolic reasoning, components of the system. Providing separate processors for the real-time and the automated reasoning components of the system prevents time-critical software components from having to compete for computation time. The choice also allows us to choose a managing processor for its symbolic computation capabilities rather than its real-time capabilities. Since the AP Executive is the only component that interacts with the robot continuously, a large scale system could probably share a single Manager among several independent work cells, each with its own AP Executive.

The Manager initiates a task by transmitting the appropriate Augmented Program (AP) to that cell's AP Executive. In return, the AP Executive transmits the event trace of the task's execution. The physical separation of functions makes this information exchange particularly important. The AP must contain all information the AP Executive requires to operate the robot in the work cell. The event trace must contain all information relating to the task's progress necessary to reconstruct activities that took place in the work cell. Whenever feasible and appropriate, the event trace contains specific numerical sensor readings from the work cell.

3. THE PREPROCESSOR

Our system uses a manipulator level robot programming language to specify the task the robot is to execute. This description is given in the AL robot programming language [Gin85b], [Muj79], though any other manipulator level language should work as well. Even though AL is not used in any commercial robot, it has many of features many languages have adopted. We have expanded AL to handle descriptions of objects so that it can be used to drive our graphic simulation system. We have chosen to use a "robot level language" to describe the task rather than a "task level language" because robots in real use are programmed with robot level languages. Starting from an existing and accepted level of language will allow us to grow to more sophisticated languages and yet to keep our ability to experiment with existing commercial robots.

We assume that the task description is accurate and correct in the sense that a robot simulator would execute it reliably. Thus the only errors the system should expect will be introduced by real world uncertainties.

We can't use the original AL program for online monitoring; we rely on an augmented version of the AL program to direct the job of monitoring the robot's activities. We call this program Augmented Program (or AP). The AP is structured as a finite state machine. The machine is represented with a directed graph in which the nodes are arbitrary states. Activities performed by the procedure are specified on the arcs connecting the states; you derive the sequence of actions in the procedure by traversing a series of arcs in the diagram. Each arc has 'events' or 'preconditions' attached to its activities. If a particular arc is leaving the state the system currently is in, then the activities on that arc are performed when the preconditions are satisfied. If two or more arcs leave a given state, then the AP Executive chooses the arc whose preconditions are satisfied first. This structure is especially useful in systems where several asynchronous events (sensor inputs) select and trigger subsequent actions.

This approach lends itself readily to the representation of AL programs. The sequencing of instructions in the original AL program is replaced by transitions in the AP. The AP representation also relates explicit sensor

data to what the robot is supposed to be doing. Crucial sensor readings preceding some action in the AL program will correspond to the preconditions of the corresponding state transition in the AP. The preconditions on arcs leading out of a given state will correspond to the set of sensor readings to be monitored by the sensor handler. Thus, the set of significant sensor readings will change automatically as the robot proceeds through its task.

Extracting high-level intentions from an AL program

Functioning of the AP Executive is highly dependent upon the information derived from the off-line portion of the system. In order for the AP Executive to interpret sensory data, the off-line component must provide the intent of the AL instructions. For example, if the intent of an instruction "move.." is to transport an object to a given location it is important to check for slippage by monitoring the touch sensors. If the intent is merely to move the arm, touch sensor data may be irrelevant. The off-line component must be able to extract the semantics of the program.

In our system methods from extracting intents of programs are based on syntactic matching and heuristics. By syntactic matching we mean identifying sequences of instructions that suggest operations such as grasping an object or releasing it. We have explored techniques of this type with excellent results. We can already handle difficult cases, such as, for instance, identifying when an object is grasped from inside a hole. Additional details can be found in [Gin85a], [Gin85c].

Figure 2 shows a state transition diagram used in conjunction with syntactic matching to identify intentions of AL instructions. Arcs shown in gray indicate transitions that have no clear interpretation and that require user intervention.

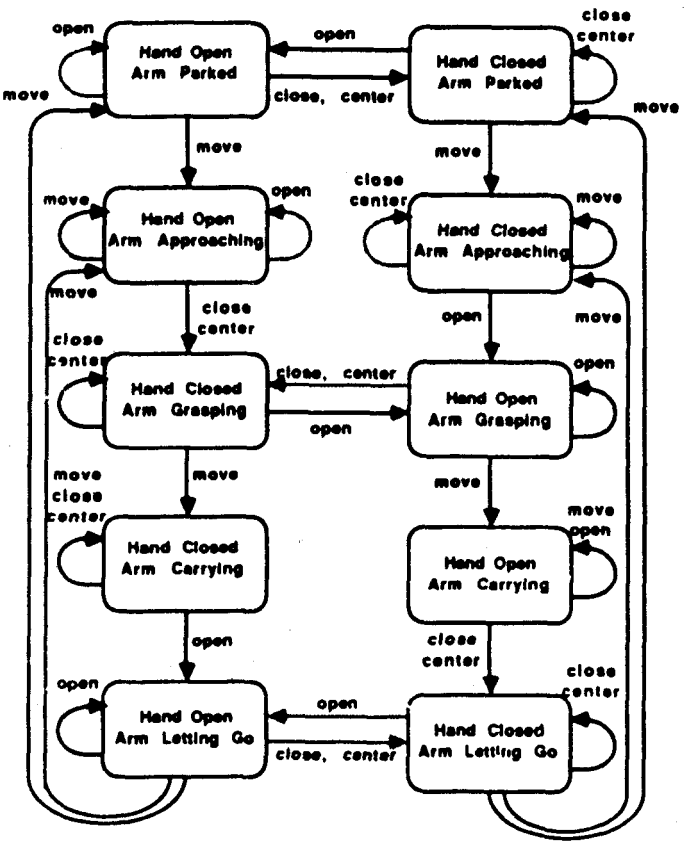

Figure 2: State Transition Diagram

Constructing the Expected World Model

The world model is another critical piece of information for the AP Executive. Suppose that a 'move object' instruction is in execution when the proximity sensors of the hand are activated. Knowledge about the

placement of objects in the environment would help determine the cause of the impending collision. In addition to placement of all objects in the environment, geometric and non-geometric properties of objects will be important. If an object being transported is slippery, the chance of dropping it is increased. If it is quite large, chances of a collision may increase, and so on.

The Preprocessor simulates the execution of the AL program to build an Expected World Model. In our current implementation the Expected World Model contains properties of the objects in the environment such as their position after the execution of each instruction, when they are moved, wich robot moves them, etc.

A classical problem with creating a world model offline is caused by branches in the program that create alternative models. Our world model has a tree structure in which branching nodes are labelled by the condition that controls the branch. The existence of the Expected World Model reduces the amount of reasoning during the recovery because we can compare the current with the expected situation to get clues about problems.

Translating an AL Program into AP.

The Preprocessor module has to provide information on how errors can be detected and what errors are likely for each instruction. For example, a 'move object' instruction might lead to a slippage error with high probability. This indicates that finger separation or touch sensors should be checked during the movement. Using the intent of the AL instructions, the Preprocessor generates in the AP program the appropriate sensory conditions to be checked to guarantee the correct execution of the instruction.

We need also to know what sensor values are relevant to detect unexpected events. Some of this knowledge is obtained from the robot program (such as the position of the robot, or the opening of the fingers), some could be obtained from the CAD data base (such as the maximum pressure to exert when grasping an object). Some values cannot be known before executing the program. Sometimes the position of an object is identified only by sensors. Hence it is important to know that no value is known in advance and that sensor data will be used.

A simple AL program and the corresponding AP are illustrated in Figure 3a and 3b. The task described in the AL program is a simple pickup operation. In the AL program WOKH indicates the robot hand and WOKA the robot arm. The example shows that we have modified AL to allow for different names of robots and different

```
OPEN WOKH TO 1.5;
MOVE WOKA TO FRAME (ROT (ROLL,-45),
 VECTOR (-7.77, -14.41, 4.4));
CENTER WOKH;
MOVE WOKA TO FRAME (ROT (ROLL,-45),
 VECTOR (-7.77, -14.41, 7));
```

Figure 3a: Example AL Program

```
((1 ((robot-do open wok 1.5))
    ((open wok) 2)
    ((hand-error wok) 12))
 (2 ((robot-do move wok (-7.77 -14.41 4.4 -45 0 0)))
    ((reach wok) 3)
    ((hit wok) 12)
    ((joint-error wok) 12))
 (3 ((imply create obj-block (-7.77 -14.41 4.4 -45 0 0))
     (expect grasp wok obj-block)
     (robot-do center wok))
    ((center wok) 4)
    ((crush wok) 12)
    ((hand-error wok) 12))
 (4 ((imply grasp wok obj-block)
     (expect carry wok obj-block)
     (robot-do move wok (-7.77 -14.41 7 -45 0 0)))
    ((reach wok) 5)
    ((hit wok) 12)
    ((untouch wok) 12)
    ((joint-error wok) 12))

        (other states in the program ... )

(12 ((imply error) (robot-do notice operator))))
```

Figure 3b: Augmented Program derived from Figure 3a

ways of expressing rotations. In the corresponding AP states are numbered. Each state contains a collection of entries. The first of them describes the action the robot has to do ("robot-do") and the meaning of the instruction in the physical world. In particular, "expect" shows what is expected to happen at the end of the state if all goes well, "imply" shows logical deductions about objects or the robot that can be made from the intentions extracted during the preprocessing phase. Each other entry specifies a condition to be checked using sensor data and the next state if the condition becomes true. Since the Preprocessor generates the conditions using knowledge about the intention of each robot action the same AL statement usually generates different conditions. As an example we can look at the states 2 and 4 in Figure 3b.

We should note that there is only one error state in the AP. When an error is detected a transition to the error state is generated. We do not use specific error states because we need to check causes of the error to find the recovery procedure. We consider the sensor data obtained as symptoms of the error not as its diagnosis. Often software that handles failures does not make the distinction between symptoms and errors or it assumes that there is a deterministic mapping between symptoms and errors.

4. THE AP EXECUTIVE

The AP Executive is responsible for maintaining an accurate picture of what the robot does. The Recoverer needs to know the robot's situation but the potential complexity of the robot's activities make it hard to derive the necessary details from the program's state. It is easiest for the AP Executive to keep track of what the robot is doing and what objects it is manipulating. The AP Executive can then provide the robot's recent history and a catalog of objects in the workspace when an error occurs.

The AP Executive does more than simply observe and report on the robot's actions. It takes responsibility for issuing commands to move the robot. When the AP says that a robot action is to occur, the AP executive sends the command to the robot. The AP Executive tracks the robot's activities by monitoring data from the robot's sensors. The sequence of sensor data yields an event trace from which we get the robot's recent history.

The Workspace Model

To recover from an error, the system needs to know what objects are in the workspace and where the objects are. At the time of error it should be easy to find out where the AL program failed and what the values of the program's variables are. Unfortunately, we can't deduce the state of the workspace from the state of the program. The program just doesn't keep the right kind of information. But it is possible to deduce when and how the AL program manipulates objects. To monitor objects in the workspace the AP Executive has to be told when an object is acquired, grasped, moved, and discarded. Typically the robot 'acquires' an object from a part dispenser, moves it somewhere, and maybe 'discards' a part by placing it on a conveyor. This is sufficient to keep track of what objects are in the workspace and where they are. The workspace model update process can then follow objects by monitoring such activities in the event trace.

The minimum workspace model is a catalog of objects and their locations. Along with the robot's most recent activities, this model gives enough information to determine what was going on at the time of an error. The Recoverer uses this information to key into more information about the object stored in the offline world model. More details about the Workspace modeller can be found in [Smi86a].

Sensor management through filtration

In the error recovery testbed, sensor information is filtered several ways. Initially, the task's AP identifies specific sensor information that is significant to the execution of that task. This information is given in terms of sensory events specified symbolically that can cause state transitions in the AP. The sensory information is used both to identify potential state transition events and to filter sensory information for the event trace. This information is also passed to "sensor filter" tasks that activate appropriate sensors and map sensor values into events significant to the progress of the robot's task.

Augmented Programs are instruction sequences structured as finite automatons. AP transitions are caused by discrete events, so a robot's progress at its task depends on the occurrence of events that cause appropriate transitions. Significant sensory readings must be mapped into events that cause state transitions. This mapping provides one form of sensory filtration: sensory readings are reported only when the value is significant to the progress of the robot's task. In some cases the identity of the event is the only specific sensory information returned and in other cases numerical data is included in the event trace as well.

Each AP state contains event predicates identifying sensor readings that would be significant to the successful execution of that state. The AP Executive passes the information in the event predicates to the appropriate sensor filters before initiating robot motion. The sensor filters activate
app
ropriate procedures so that necessary sensor readings will take place. The AP Executive omits all sensor information from the event trace that is not identified in the AP as being significant.

Grip sensor filtering on the IBM 7565 is implemented using "monitor" facility of the AML language [Tay82]. Monitors are used to define ranges of sensor values that can activate user-defined procedures or terminate robot motions. When initializing the IBM 7565, the AP Executive defines a set of monitors for classifying gripping forces and associates each monitor with an AP event type. The numerical values used for classifying gripping force depend on the objects being used in the robot's task and the actions performed on them, so these values may be adjusted during initialization.

When the AP Executive gives the IBM 7565 a motion command, it also specifies a set of monitors to activate. The AML system collects the appropriate sensor readings for each active monitor and "trips" the
appropriate monitor if its sensor enters the monitor's defined range. This terminates the motion in progress and generates a message to the AP Executive identifying the qualitative value of the sensor reading, as determined from the monitor that was tripped. If no monitor terminates the active motion, a similar message indicating uninterrupted completion is sent instead. The AP Executive then generates a sensor event and, if necessary, updates the event trace and performs an AP state transition.

Since the IBM 7565 is normally programmed in the AML language, the AP Executive translates AP commands and sensor specifications into an AML compatible form. Since AML is a manipulator level language, the mapping of robot actions from the AP form is straightforward. Mapping sensor operations is more complex since APs specify symbolic sensor readings. Figure 4 shows an example of the transformation of a "move" operation in an AP state into the corresponding AML commands executed by the robot. The desired destination and the desired AML monitoring sets to be activated (E_HIT and E_UNTOUCH) are passed to the APM procedure. This procedure, written in AML and executing on the IBM series 1, performs the MOVE operation and the related filtering for the 7565's sensors. The procedure activates the appropriate monitors and performs the motion subject to the selected monitors.

Qualitative sensor interpretation

Although the event trace often provides numerical sensor data, such information is not of primary importance when reasoning about the robot's activities. To meet this need, the error recovery system assigns symbolic meanings to numerical sensor values in a number of ways. Spatial locations and critical dimensions are assigned symbolic names. Gripping forces are assigned qualitative values according to the range in which a force value falls.

Qualitative classification of sensor data often serves a second purpose as well. When executing an AP, the AP Executive responds to events in terms of symbolic classifications. Upon successful completion of a motion command the AP responds to a "reach" sensor event instead of examining and matching the robot's reported destination. If the gripper drops an object and the gripping force drops to a small value, the AP responds to an "untouch" sensor event instead of testing the specific force value. The classification of sensor values into different types of AP events is performed by a sensor filter procedure that operates on the behalf of the AP Executive, as described above.

Ideally, qualitative classification of sensor data should be performed by a component of the Manager and exploit its increased computational capabilities and knowledge bases. Symbolic classification of spatial information is an example of this. The robot's sensor filter identifies whether successful completion of a robot motion caused the robot to reach its sensed position, but the filter does not try to identify the location in terms of the robot task's overall goals. Identification of the particular location is handled by the Manager's work cell modeller.

When an error occurs the Manager produces a model of the robot work cell in terms of its probable state and its intended state. Locations visited by the robot or by objects in the work cell are assigned symbolic identifiers, and a history is produced of visits for each location, object, and robot gripper in the work cell. Error recovery planning consists of producing a sequence of robot actions to change the work cell from its erroneous state to its intended state.

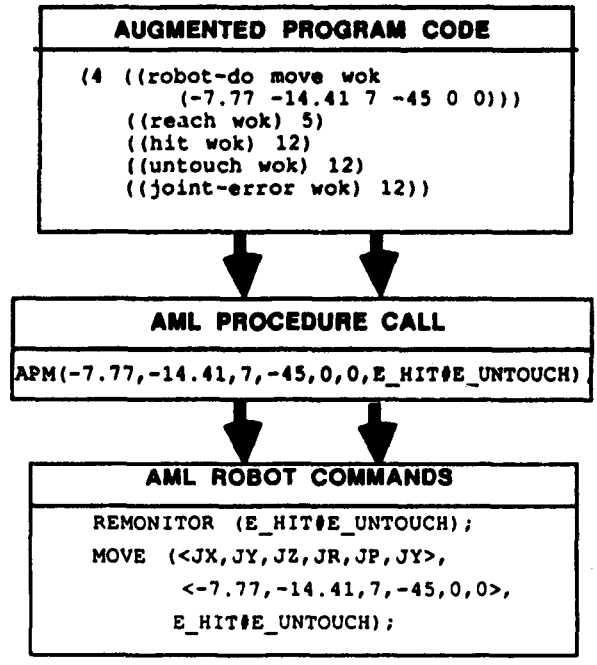

Figure 4: Converting from AP statements to AML statements

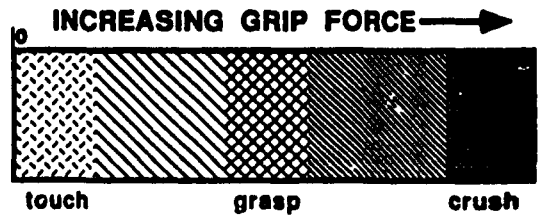

Figure 5: Qualitative interpretation of grasping forces

Qualitative interpretation of gripper forces, on the other hand, must be performed by a sensor filter. Gripper forces, when they are significant, determine whether the gripper is touching an object and holding with an adequate force. Identification of appropriate touching and grasping forces must be communicated to the AP Executive so that appropriate AP state transitions occur depending on the gripping forces encountered. The sensor filter classifies gripping forces into specific ranges according to the robot's current action. Each range corresponds to a type of sensing event that can be produced by the gripping force sensor. Figure 5 shows an example of that.

5. THE RECOVERER

We are interested in discovering causes for errors and errors might appear a long time after what caused them happened [Smi86b]. Error interpretation becomes more difficult as the complexity of the task increases. For example, consider a task where a robot moves cubes from a feeder to a shipping pallet, twelve at a time. What might happen if the the cube falls from the gripper and lands on the pallet, knocking another object off? Most of the failure reason models available only apply to the objects and situations directly related to the sensor reading indicating the error. The robot thus only associates an error with a part if it uses its sensors on the part and finds an error. The lost part won't be missed until someone down the line tries to unload the pallet and finds it one part short.

The symbolic model of the work cell constructed before execution of the task and the trace of events are used here. When something unexpected happens we can trace the error back until we find critical measurements or assumptions made that were not supported by sensor data. For instance, in general we assume that if something is left in a stable configuration it will remain there until a fact appears that shows the configuration has changed. If we discover later that the object moved it means that something happened to move it that was not explicitly noted.

We have developed methods for symbolic tracking of objects from event traces. Common sense heuristics are used for symbolic tracking. For instance, if the robot is holding an object and the robot moves, then we know that the object moves to the same place. Tracking an object in real time with sensors is too expensive, unless we know where to look for it, and how it looks like. Symbolic tracking is less expensive from a

computational point of view, and helps in reducing the number of possible causes for errors. After that number has been reduced we can get additional sensor data to guarantee that the correct cause for the error has been identified.

It is curious to observe that most of the AI work in planning has concentrated on checking preconditions before executing every action to guarantee that they are satisfied in the current state of the world. Postconditions are not checked, but are used solely to update the world model. So an error can go unchecked for a while and can be detected only when it affects the preconditions of another action. We check selected conditions after the execution of each instruction to guarantee that failures are identified as soon as they appear. This still does not solve the problem of errors caused by the robot during the movement that require different sensors to be checked. For instance, if the arm bumps into objects during a transfer motion without losing the part it is carrying no error will be reported. This requires failure reason analysis when an error is found.

Once an error has been detected, the recovery process can start using a trace of relevant events and whatever information is available about the task to determine the causes and effects of the error. It is only after the cause for the error has been identified or, at least, after alternative plausible causes has been found the recovery process can start.

To be more specific, if an AP state transition leads to an error state, a message to that effect is appended to the event trace and the trace is passed to the Recoverer component of the Manager. The Recoverer generates a model of the current work cell's state and of its desired state. This model is used to produce a recovery plan in the form of AP states to be appended to the task's existing AP. The Manager passes these additional states back to the AP Executive where they are executed. If the new states each execute successfully, they will lead the task back to a state in the original AP.

To successfully effect recoveries in this manner, the Recoverer requires a copy of the task's AP and the information in the event trace. The Recoverer can also exploit knowledge about the robot's task, the parts involved, and the work cell to produce the recovery plan. To simplify experiments with error recovery as well as for improved performance in industrial situations, the testbed's AP Executive can handle repeated failures and subsequent recoveries by a robot task. The AP Executive can also display messages on the AP Executive's display screen for explaining error diagnoses or for instructing the robot's operator.

During normal execution, the AP Executive contains a copy of the robot task's AP. If an error recovery occurs, the Recoverer passes additional AP states to the AP Executive. These additional states do not replace existing states in the AP; they are appended to them. To recover, the AP Executive resumes task execution with the first of recovery states passed to it. Once the recovery execution begins, the AP Executive treats the recovery states identically to the states in the original AP. If another failure occurs, whether during the recovery or after completing the recovery, the AP Executive again reports the failure to the Recoverer and resumes execution when it receives a set of recovery states.

For example, if the robot loses a part, it can attempt a recovery by opening the gripper, moving to the work cell surface, and trying to grab the part. If the part is there, the recovery can proceed. If the grasp fails, the AP Executive simply informs the Recoverer which can then produce another recovery plan and try again.

The ability to do multiple recoveries allows the Recoverer to profit from mistakes in a recovery plan. When faced with multiple recovery choices, the Recoverer can choose the one that is most likely to reduce uncertainty about the state of the work cell. The Recoverer can also produce recovery plans with the sole purpose of taking sensor readings in the robot work cell. If the Recoverer needs to probe a specific spatial location it can produce a recovery plan that performs the desired sensor reading and then immediately fails. The resulting event traces will increase the amount of information in the work cell model and the unsuccessful recovery will not prevent a subsequent recovery from being attempted.

Another useful feature during error recovery is the AP Executive's ability to display messages for the robot's operator. These messages are produced by statements in the AP and thus may be generated by the Recoverer. This facility allows the Recoverer to request specific operator intervention when necessary. This permits experiments with failures that tax the available sensory or reasoning facilities. In the Testbed it also allows experiments with primitive Recoverer software that simply diagnoses the problem and asks for operator intervention. This capability may also have worthwhile industrial applications: the Recoverer could produce messages to guide the robot's operator in manually correcting problems in a complex and unfamiliar assembly. An example is shown in Figure 6.

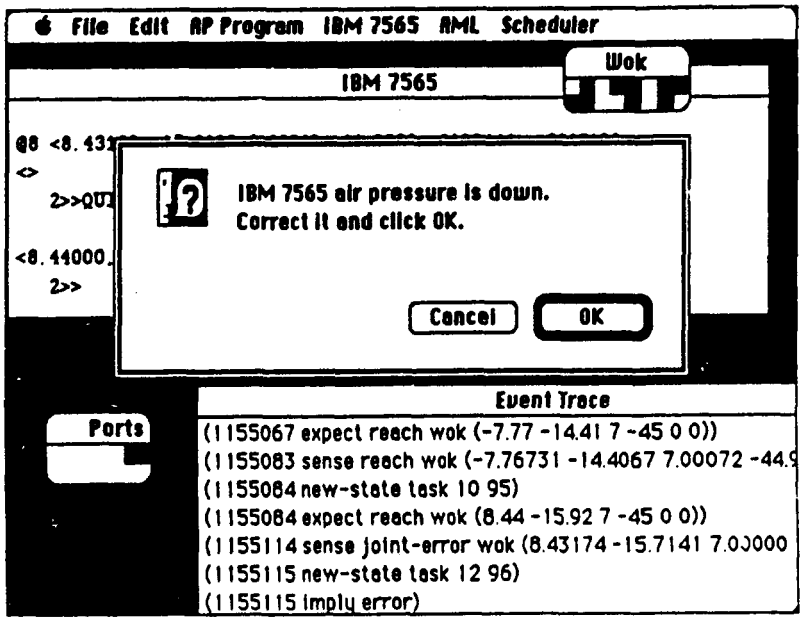

Figure 6: Operator display by AP Executive

6. RELATED WORK AND SUMMARY

Our approach differs from other approaches in significant aspects. The current state of the art in industrial robots is that either the robot executes its task regardless of its success or it quits every time it encounters something unexpected [Luh83]. A better approach is to handle the situation by preprogramming error checking and error recovery procedures for every probable error [Bon82], [Gin83a], [Gin85b], [Tay82] This is an expensive method both in engineering and in robot computational resources. It is also easy to forget some important checks.

Since it is difficult to consider all possible errors, many of which might never happen, another method is to generate from the task level description a program that is guaranteed to be correctly executed even in the presence of uncertainties in the environment. This requires models of robot kinematics and dynamics, and models of physical properties of objects such as friction. This approach has been applied only to fine motions for specific tasks such as insertion operations [Loz84]. Modeling uncertainties [Bro82] and taking into account errors in the model [Don86] helps but the real world is so complex that it might not be worth developing sophisticated models of it.

Much previous research in Artificial Intelligence has centered on detection and correction of errors in simulated robot systems [Wil84]. These studies all make a number of assumptions: knowledge about events is correct, each action produces precisely defined postconditions, there are no uncertain data, correct predicates are generated from sensor data every time they are needed, and sufficient knowledge is provided to take into account all the possible states of the environment. These assumptions are too strict to be realistic.

A few exceptions exist. The most notable is STRIPS [Fik71] the system used to control the mobile robot Shakey. Srinivas [Sri76] [Fri77] has designed a system for analyzing the causes of failures in robot programs and for replanning the robot activity. More recently, work has been done on monitoring the execution of programs with real robots [Lee83], [Lop86]. There is a growing interest in modeling sensors [Fox83], [Hen84], and planning for their use [Doy86] that will provide needed background for work on error detection and recovery.

Our approach is more similar to the way people handle errors and unexpected events. By relating events to general knowledge human beings can identify unexpected situations and by applying common sense and domain specific knowledge they can find solutions to situations never seen before. The key to human performance is in the knowledge about the environment and about the specific task at hand. We want to do something similar for assembly robots. Since the domain is limited and reasonably constrained the amount of knowledge needed can be managed by using present technology [CAR84].

ACKNOWLEDGEMENTS

Several individuals have contributed to the research presented in this paper. We would like to acknowledge the contribution of Rajkumar Doshi, Sharon Garber, Marc Gluch, Shideh Hojat, and Imran Zualkernan.

This work was funded in part by NSF grant DMC-8518735, by the Microelectronic and Information Sciences Center and by the Productivity Center of the University of Minnesota.

REFERENCES

[Bon82] Bonner, S., and Shin, K., "A comparative study of robot languages," Computer Magaz., December 1982, pp 82-96.

[Bro82] Brooks, R. A., "Symbolic Error Analysis and Robot Planning," International Journal of Robotics Research, Vol 1, N 4, Winter 1982, pp 29-68.

[CAR84] Committee on Army Robotics and Artificial Intelligence, et al, "Applications of Robotics and Artificial Intelligence to reduce risk and improve effectiveness," Robotics and Computer-Integrated Manufacturing, Vol 1, N 2, pp 191-222, 1984.

[Car85] Carlisle, B., "Key Issues of Robotics Research", in Hanafusa, H., and Inoue, H. (eds) Robotics Research, The Second International Symposium, The MIT Press, Cambridge, Mass, 1985, pp 501-503.

[Don86] Donald, B., "Robot Motion Planning with Uncertainty in the Geometric Models of the Robot and Environment: a Formal Framework for Error Detection and Recovery", Proc 1986 IEEE Conference on Robotics and Automation, pp 1588-1593, San Francisco, April 1986.

[Doy86] Doyle, R.J., Atkinson, D.J., and Doshi, R.S., "Generating Perception Requests and Expectations to Verify the Execution of Plans", Proc. AAAI-86, Philadelphia, August 1986, pp 81-87.

[Fik71] R. E. Fikes, and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving", Artificial Intelligence 2, pp 189-208, 1971.

[Fox83] Fox, M.S., et al., "Techniques for Sensor-Based Diagnosis", Proc IJCAI 83, pp 158-163.

[Fri77] Friedman, L., "Robot learning and error correction," Proc. 5th International Joint Conference on Artificial Intelligence, pp 736, 1977.

[Gin83a] Gini, G., and Gini, M., "Explicit programming languages in industrial robots", Journal of Manufacturing Systems, Vol 2, N 1, 1983.

[Gin83b] Gini, M., Gini, G., "Towards automatic error recovery in robot programs," Proc. 8th International Joint Conference on Artificial Intelligence, August 1983, pp 821-823.

[Gin85a] M. Gini, R. Doshi, M. Gluch, R. Smith, I. Zualkernan, "The role of knowledge in the Architecture of a Robust Robot Control," Proc. 1985 IEEE International Joint Conference on Robotics and Automation, pp 561-567, March 1985, St. Louis, Missouri.

[Gin85b] Gini,G. , Gini, M., "Dealing with world model based languages," ACM Trans on Programming Languages, Vol 7, N 2, April 1985, pp 334-347.

[Gin85c] M. Gini, R. Doshi, S. Garber, M. Gluch, R. Smith, I. Zualkernan, "Symbolic Reasoning as a Basis for Automatic Error Recovery in Robotics," Technical Report TR 85-24, Computer Science Department, University of Minnesota, August 1985.

[Hen84] T. Henderson, E. Shilčrat, "Logical Sensor Systems", Journal of Robotics, 1, 2, pp 169-193, 1984.

[Lee83] M. H. Lee, D. P. Barnes, N. W. Hardy, "Knowledge based error recovery in industrial robots", Proc. 8th International Joint Conference on Artificial Intelligence, August 1983, pp 824-826.

[Lop86] E. Lopez-Mellado, R. Alami, "An execution monitoring system for a flexible assembly workcell", Proc. 16th ISIR, September 1986, pp 955-962.

[Loz83] Lozano-Perez, T. "Robot Programming," Proc. of the IEEE, Vol 71, N. 7, July 1983, pp 821-841.

[Loz84] Lozano-Perez, T., Mason, M.T., Taylor, R.H, "Automatic Synthesis of Fine-Motion Strategies for Robots," The International Journal of Robotics Research, Vol 3, N 1, Spring 1984, pp 3-24.

[Luh83] Luh, J. Y. S., "An anatomy of industrial robots and their controls," IEEE Trans. on Automatic Control, Vol AC-28, N. 2, 1983.

[Muj79] Mujtaba, M.S. and Goldman, A., "AL users' Manual", Stanford Artificial Intelligence Laboratory Memo AIM-323, Stanford, Ca, January 1979.

[Smi86a] R. Smith, M. Gini, "Robot Tracking and Control Issues in an Intelligent Recovery System" Proc 1986 IEEE Conference on Robotics and Automation, pp 1070-1075, San Francisco, April 1986.

[Smi86b] R. Smith, M. Gini, "Reliable Real-time Robot Operation Employing Intelligent Forward Recovery," Journal of Robotic Systems, Summer 1986, pp 286-301.

[Sri76] Srinivas, S., "Error recovery in a robot system," PhD Thesis, CIT, 1976.

[Tay82] Taylor, R.H., Summers, P. D., Meyer, J. M., "AML: a manufacturing language," International Journal of Robotics Research, Vol 1, N. 3, 1982.

[Wil84] Wilkins, D., "Monitoring the execution of plans in SIPE", Computational Intelligence, Vol 1, 1985, pp 33 45.